Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

FMPS on MIC

Florian Seybold[a]a*, Ralf Schneider[a], David Horák[b], Lubomír Říha[b], Václav Hapla[b], Vít Vondrák[b]

[a]*High Performance Computing Center Stuttgart (HLRS)*
[b]*IT4Innovations, VŠB-Technical University of Ostrava (VSB)*

Abstract

The Finite Element Method software system FMPS is briefly introduced and an overview is given of development work done by HLRS and VSB concerning the acceleration of solutions to problems yielded by FMPS using Intel's Many Integrated Core architecture.
HLRS implemented a hybrid MPI/OpenMP Conjugate Gradient solver. A different scaling behaviour when using Intel Xeon Phi cards due to a different message transportation (involving a PCIe 2.0 bus) is revealed.
VSB shows that the number of iterations of a Conjugate Gradient solver can be significantly reduced by using a Deflated Conjugate Gradient scheme, involving the need for an additional solution to a dense linear system by a LU factorisation. The solution to the dense system can be accelerated using MIC cards.

1. Introduction

The Finite Method Programming System (FMPS) is a modular software system, which can be used for the simulation of large deformation coupled thermomechanic processes using the Finite Element Method (FEM). Applications of FMPS include

- quasistatic large deformation analysis of elastic-plastic solids,
- large deformation analysis of viscoplastic solids and
- instationary heat transfer.

It is used by academic institutions including HLRS as well as consulting engineers.

In this project two groups of PRACE experts worked on the parallelisation for a distributed Many Integrated Core (MIC) card system of the linear solver part:

- HLRS implemented a hybrid MPI/OpenMP parallel Conjugate Gradient (CG) method using the Sliced ELLPACK (SELLPACK) format for the sparse matrix description,
- VSB implemented a MPI parallel Deflated Conjugated Gradient (DCG) method which interfaces FMPS via the PETSc library.

* Corresponding author. *E-mail address*: seybold@hlrs.de

2. Hybrid MPI/OpenMP parallel CG method for Intel Xeon Phi systems

A performance critical operation within the CG method used to solve FEM problems is the sparse-matrix-vector-multiplication. Especially many core architectures like Intel Xeon Phi or NVIDIA GPUs with low overall cache sizes may benefit from a sparse matrix representation, which is optimised for data access by a matrix-vector-multiplication-routine. A suitable sparse-matrix-representation for the Intel MIC architecture is the SELLPACK-format by Alexander Monakov et al. [1].
The SELLPACK format slices the rows in heights which suits the vector operation register width of the architecture and stores the data in a slice within a column-major format. Additionally the rows in a slice are padded to the largest row length in this slice in order to facilitate the access of slice data with a minimum of cache line loading and storing operations. On the other hand, the padding results in increased memory consumption.

In the first step, a shared memory implementation of the CG method using Fortran and OpenMP has been implemented and optimised for the Intel Xeon Phi architecture. It is worth noting that the very same implementation for the Intel Xeon Phi can also be used for the Intel Sandy Bridge like CPU architectures, concerning both the compilation of the source code as well as the optimisation work done in the source code for the Intel Xeon Phi architecture. This makes the development of code for both Intel Xeon Phi and Intel Sandy Bridge quite convenient.
Figure 1 shows the performance and memory bandwidth of the shared memory implementation with respect to different degrees of freedom (DOFs) of a discrete Poisson matrix problem on an Intel Xeon Phi card and a Sandy Bridge eight core CPU.

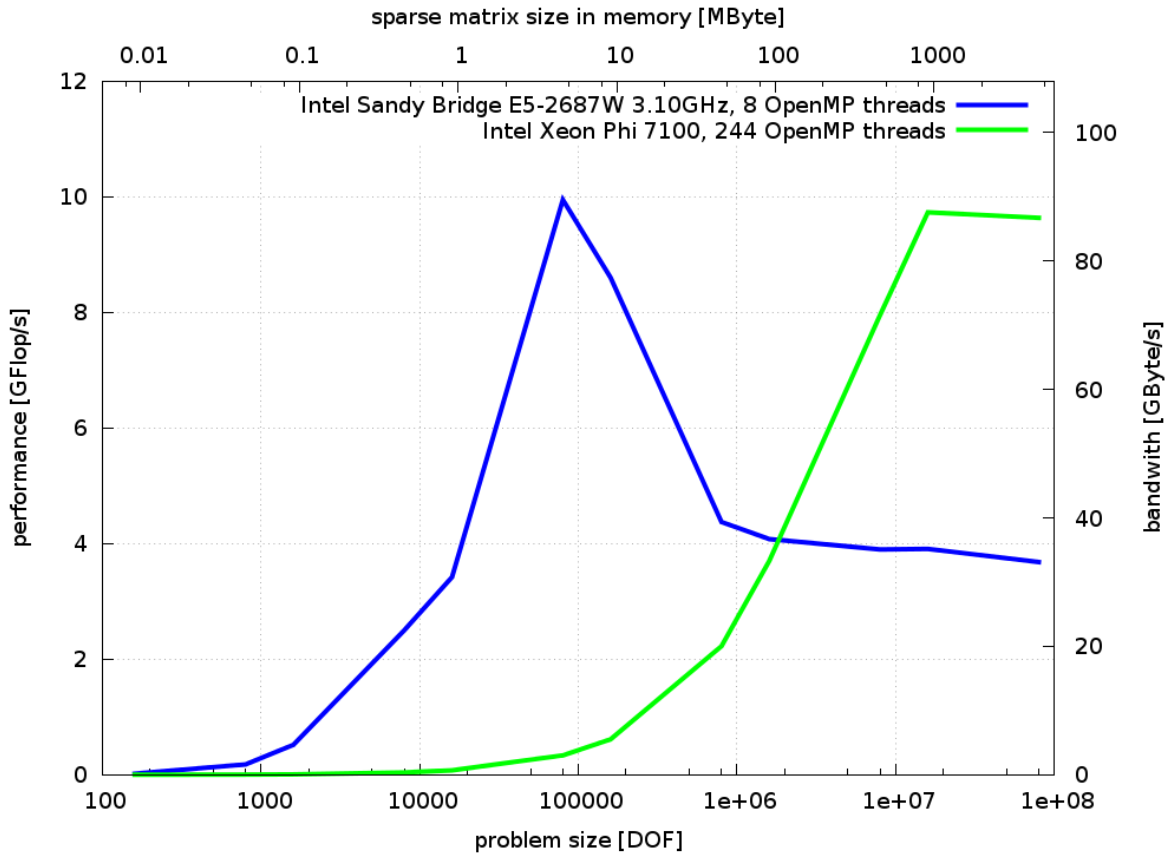

Figure 1: OpenMP parallel Conjugate Gradient method implementation solving sparse 2D Poisson matrix problems of different sizes using double precision. A 2D Poisson matrix comes with an average of five elements per row.

One can clearly see a cache effect at around 100,000 DOFs on the CPU, but not on the Xeon Phi card. The Xeon Phi shows clearly better performance than the CPU at around ten million DOFs upwards, although it seems to saturate at only 27% of the peak bandwidth of 350 GB/s, while the Sandy Bridge CPU shows 68% of its peak bandwidth of 51 GB/s at hundred million DOFs.

In the next step, the shared memory implementation of the CG method has been MPI parallelised for a distributed memory system, yielding a hybrid MPI/OpenMP implementation of the CG solver. The MPI

parallelisation has been implemented in a naive way, i.e. each part of the matrix-vector-multiplication operand vector of a local node is sent to all nodes. A smarter implementation would identify unnecessary communication of vector parts and furthermore permute the DOFs, for example by the Cuthill–McKee[2] algorithm, in order to increase the number of vector parts, which don't have to be communicated.
Figure 2 shows the strong scaling and Figure 3 shows the performance of the hybrid MPI/OpenMP solver implementation working on a problem formulated by FMPS with a fixed number of DOFs, measured on different machines/configurations:

- Cray XC30 machine with Sandy Bridge nodes, one MPI process per NUMA node, eight OMP threads per MPI process.
- Eurora system with Sandy Bridge nodes, one MPI process per NUMA node, eight OMP threads per MPI process.
- Eurora system with Intel Xeon Phi cards, one MPI process per card, 236 OMP threads per MPI process.

The number of 236 OMP threads has been chosen under the assumption, that one core with four threads should be spared for processing MPI communication, as it should be done in offload mode, see Weinberg et al. [3], section 5.1.2. On the one hand, this assumption may be false; on the other hand, there shouldn't be much difference in performance between the usages of 236 or 240 threads.
Note that a node consists of two Sandy Bridge processors (NUMA nodes) with eight cores per processor at the Cray XC30 and Eurora and additional two Xeon Phi cards at Eurora.

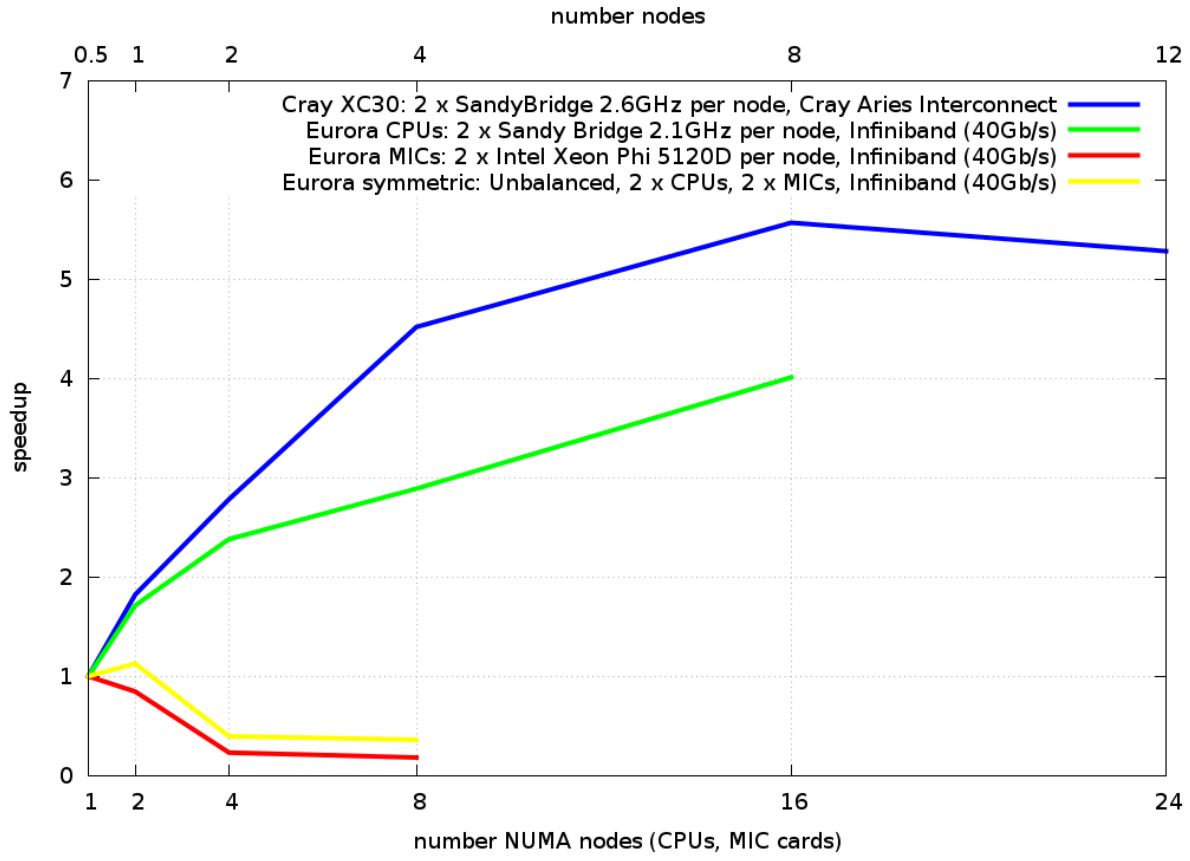

Figure 2: Strong scaling of the hybrid MPI/OpenMP Conjugate Gradient solver with a FMPS problem size of 185610 degrees of freedom.

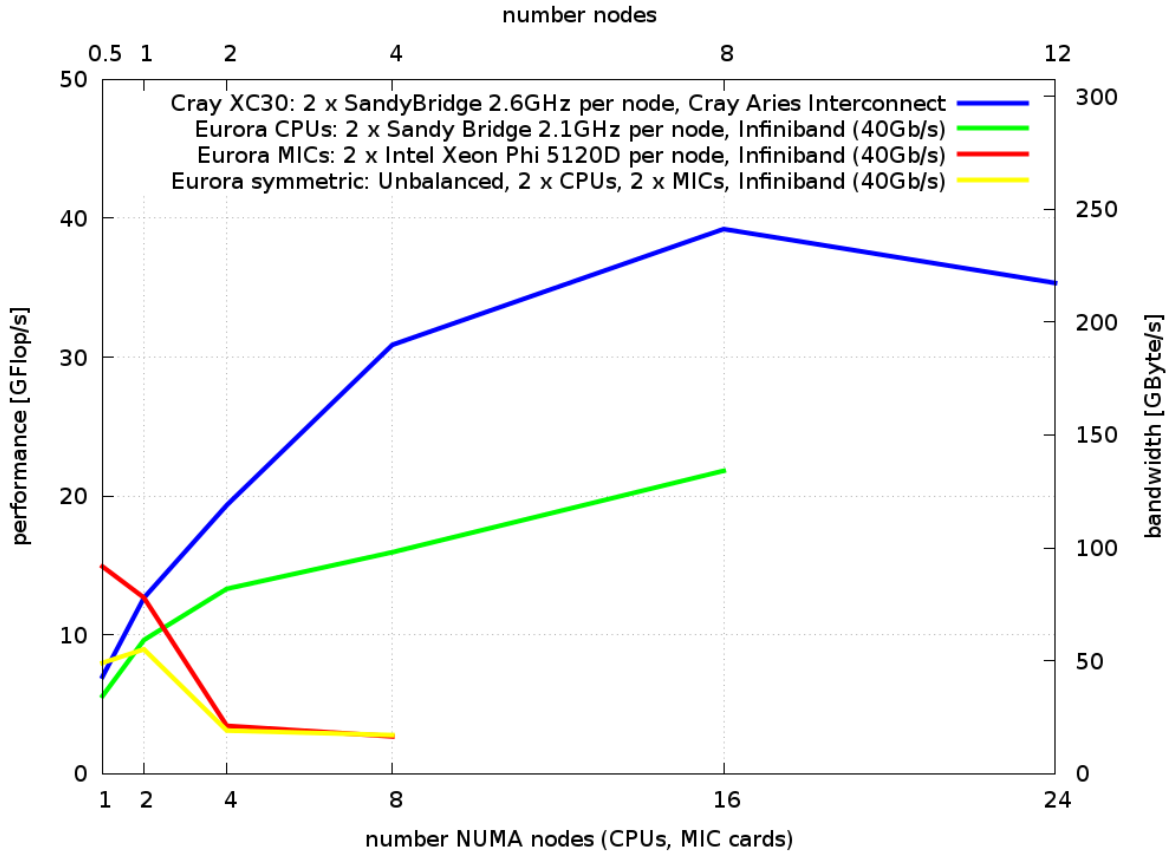

Figure 3: Projected performance of the hybrid OpenMP/MPI Conjugate Gradient solver with a FMPS problem size of 185610 degrees of freedom.

The Intel MPI library is used in conjunction with the following OpenMP and MPI specific environment:

```
export I_MPI_MIC=enable
export I_MPI_DAPL_PROVIDER_LIST=ofa-v2-mlx4_0-1,ofa-v2-scif0,ofa-v2-mlx4_0-1
export I_MPI_PIN_DOMAIN=omp
export I_MPI_ENV_PREFIX_LIST=snb:SNB,knc:KNC
export SNB_KMP_AFFINITY=scatter
export KNC_KMP_AFFINITY=scatter
export SNB_OMP_NUM_THREADS=8
export KNC_OMP_NUM_THREADS=236
```

In general, the mild speedup of the Cray XC30 and the Eurora system with CPUs only and the bad (below one) speedup of the Eurora system with Intel Xeon Phis suggests a communication intensive MPI implementation. The most interesting issue seems to be the different speedup behaviour within the Eurora system when using CPUs only and when using Xeon Phis only, or CPUs and Xeon Phis in a symmetric mode. In Figure 3, one can see more than double performance of one Xeon Phi card compared to one CPU, which drops when using more Xeon Phi cards, while the CPU only performance increases.
This behaviour suggests different communication costs with regard to different architectures (CPU and MIC). Figure 4 shows round-trip latency measurements of a MPI ping-pong test with different message sizes, conducted with pairs of CPU – CPU, CPU – MIC, MIC – MIC within the same node and over different nodes.

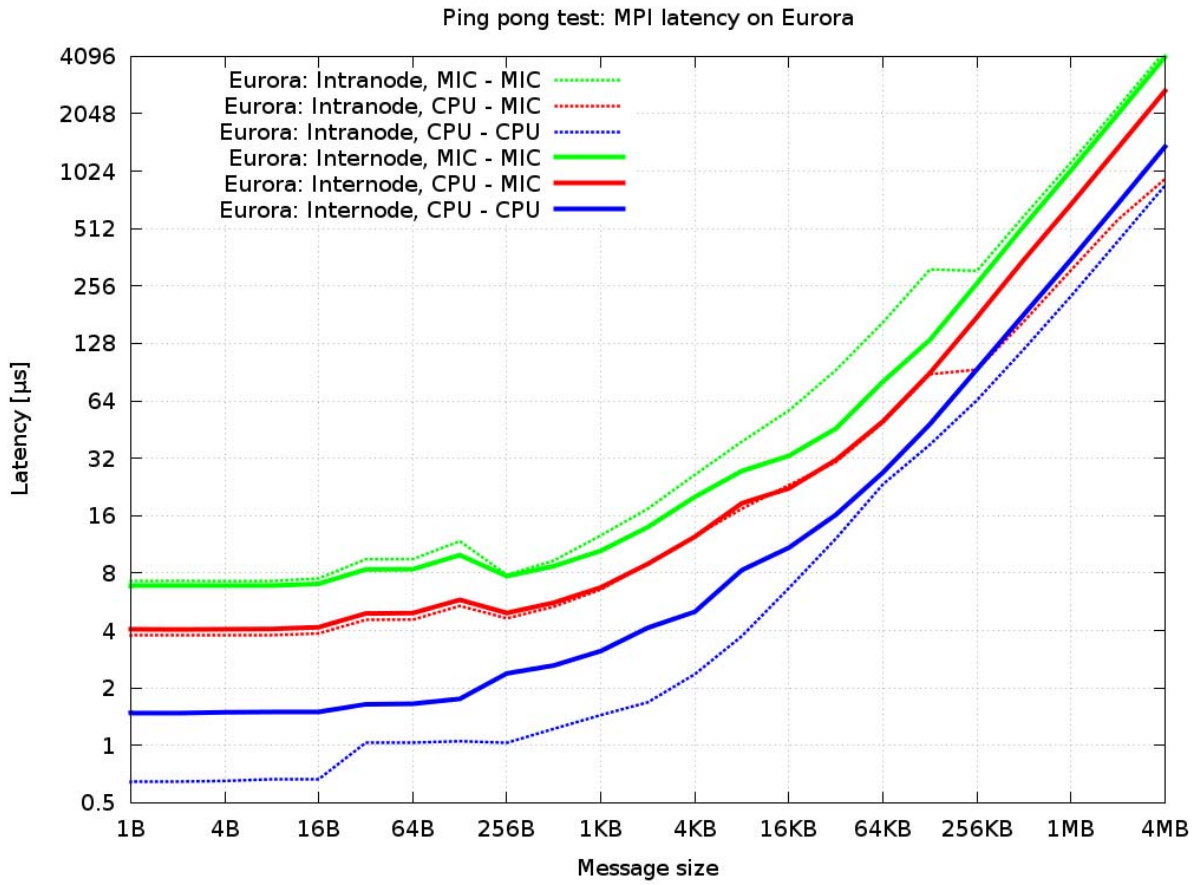

Figure 4: MPI Ping-pong test with different message sizes.

In Figure 4 we see in general a constant ping-pong latency time to a message size of 16 Byte, which then changes into a nearly linear function with respect to the message size after a message size of about 256 KByte. This indicates communication costs bound to a base message latency for small messages and bound to a transport bandwidth for larger messages.
If we model the data in Figure 4 as the bandwidth of the communication, for a message size of 4 MByte, we would see a communication bandwidth of about 1 GByte/s between MIC cards within the same node or over different nodes, and a communication bandwidth of about 3 GByte/s between CPUs over different nodes, and 4.8 GByte/s between CPUs within the same node.

3. The adaptation of parallel DCG method for Intel Xeon Phi systems

Discretization of most engineering problems, leads to large sparse linear systems of equations $Ax = b$. These systems can be then solved by means of **direct, iterative** or **hybrid** (combining both previous) **solvers**, which can be also categorized into **primal, mixed** (saddle point) or **dual** methods [8]**.**

When the sequential solution is not possible (because of the large problem's size, memory requirements or long computational times), it is necessary to solve it in parallel. Parallelization of efficient algorithms for the solution of linear systems can be implemented mostly using SPMD technique **– distributing data** among Nc processing units

$$A = \begin{bmatrix} A^1 \\ \vdots \\ A^{Nc} \end{bmatrix}, b = \begin{bmatrix} b^1 \\ \vdots \\ b^{Nc} \end{bmatrix}.$$

This allows to use almost identical algorithms for both sequential and parallel case; only data structure implementation differs.

The basic advantage of the **primal methods** is smaller memory requirements and low cost of iterations. But a large number of iterations needs to be performed as described by the well-known estimate for the convergence of CG method in *k*-th iteration:

$$\|r_k\| = \|b - Ax_k\| \leq 2\sqrt{\varkappa}\left(\frac{\sqrt{\varkappa}-1}{\sqrt{\varkappa}+1}\right)^k \|b\|.$$

The main objective of this subtask being the joint work of HLRS and VSB is to reduce the number of CG iterations by means of simple deflation using aggregations based on domain decomposition – so called deflated CG (DCG) method. The implementation of DCG is designed to run on MIC accelerated systems.

The error is propagated globally by means of an additional coarse problem in form

$$W^T AWy = W^T Ar,$$

where the columns of the matrix W are a basis for the deflation subspace. For larger coarse problems then e.g. SuperLU_Dist [7] parallel solver running on subcommunicators can be used. The simplest way of defining this mapping W for a mesh-based system of equations is by agglomerating the nodes of the mesh into subdomains and then defining a polynomial reconstruction over each of them. In our case the entries in W are unity for the DOFs of this region, and zero for all other DOFs. The DOFs with Dirichlet boundary conditions have to be excluded from the W construction. At the algorithmic level it can be viewed as subtracting one more (low-dimensional) search direction Wy from the original CG search direction, so that $p = r - \beta p - Wy$.

CG/DCG algorithms: initialize: $x = 0, r = b - Ax, p = r$

while $\|r\|/\|b\| > tol$

$$\alpha = \frac{r^T p}{p^T Ap}$$

$$x = x + \alpha p$$

$$r = r - \alpha Ap$$

$$\beta = \frac{r^T Ap}{p^T Ap}$$

$$p = r - \beta p \quad \boxed{-Wy, \qquad W^T AWy = W^T Ar}$$

end

The numerical experiments were run on supercomputers

- HECToR at EPCC, the Phase 3 system is contained in 30 cabinets and comprise of a total of 704 compute blades. Each blade contains four compute nodes giving a total of 2816 compute nodes, each with two 16-core AMD Opteron 2.3GHz Interlagos processors. This amounts to a total of 90,112 cores. Each 16-core socket is coupled with a Cray Gemini routing and communications chip. Each 16-core processor shares 16 GB of memory. The theoretical peak performance of the phase 3 system is over 800 Tflops.
- Anselm at VSB, 207 compute nodes, each node is equipped with two eight-core x86-64 Sandy Bridge processors,180 nodes without acceleration, 23 nodes with GPU accelerator (Nvidia Tesla M2090), 4 nodes with MIC accelerator (Intel Xeon Phi P1640), 64GB RAM and a 300GB local disk, the peak performance is 82 Tflops.

For the numerical testing an elastic car engine model was discretized to 2.5M DOFs and partitioned into 102 or 1014 subdomains with Metis. The DCG method performs significantly better than CG. Benchmarks were computed on Cray HECToR at EPCC.

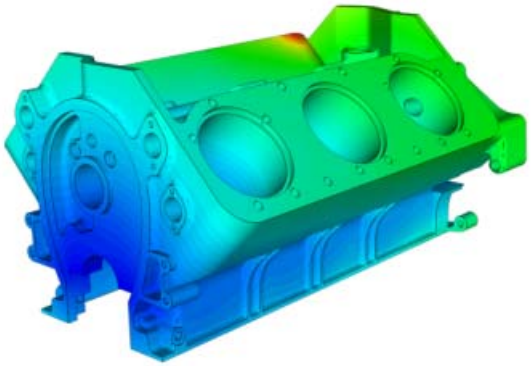

Figure 5: Car engine benchmark

Nc	Iters/Time	CG	DCG
102	Iters	**14,962**	**2,402**
	Preproc, time	0.025	37.2
	Solution time	1,005.0	484.6
	Total time	**1,006.0**	**533.1**
1,014	Iters	**16,021**	**2,693**
	Preproc. time	0.013	49.7
	Solution time	976.0	513.2
	Total time	**977.0**	**565.8**

Table 1: Comparison of CG and DCG methods on HECToR (coarse problem solved using SuperLU)

The performance of the coarse problem solution can be increased using GPU or MIC accelerators. The efficiency of the direct solution of a dense coarse problem using Magma LU solver [6] on CPU only (16 OpenMP threads), CPU+GPU (16 OpenMP threads + 1 GPU), CPU+MIC (16 OpenMP threads + 1 MIC, 240 OpenMP threads) if executed on the single node is demonstrated by the following table and graphs. Benchmarks were computed on the Bull cluster Anselm at VSB.

		CPU only		**CPU+GPU**		**CPU+MIC**	
	size	fact	solve	fact	solve	Fact	solve
1.	384	0.002	0.0001	0.004	0.0006	0.002	0.0012
2.	3,072	0.125	0.0046	0.075	0.0187	0.123	0.0037
3.	6,000	0.769	0.0190	0.267	0.0432	0.374	0.0086
4.	24,576	46.066	0.2329	11.349	0.2275	22.495	0.0829

Table 2: Comparison of Magma LU dense solver (factorization/solve) for various coarse problem sizes on CPU only, CPU+GPU and CPU+MIC on Anselm

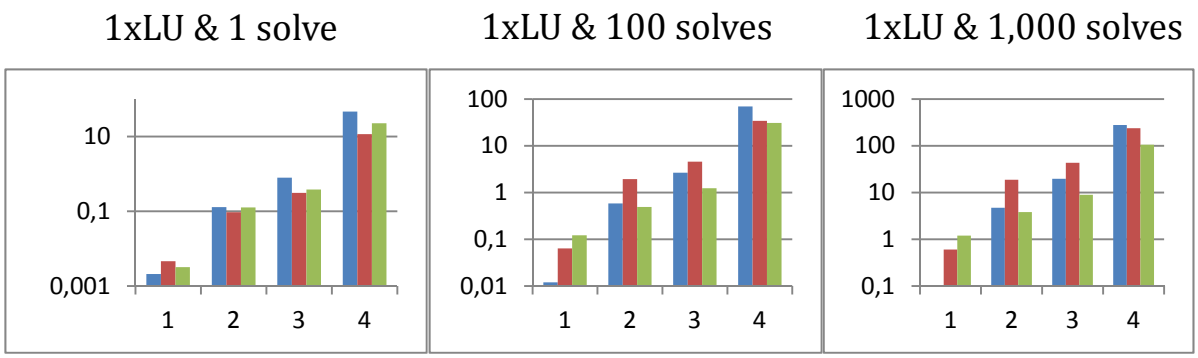

Figure 6: Performance comparison of the CPU only (blue), CPU+GPU (red) and CPU+MIC (green)implementations on various coarse problem sizes(384; 3,072; 6,000; 24,576) on Anselm

We can see that the LU factorization using CPU+MIC is 2 times faster and that using CPU+GPU even 4 times faster than one using CPU only. For large number of solves, e.g. 1,000, the time for CPU+MIC is 82.9 sec, for CPU+GPU 227.5 sec and for CPU only 232.9 sec, so CPU+MIC is 3 times faster than CPU only or CPU+GPU.

From the implementation point of view, we have incorporated DCG method as a new KSP type, which can be specified by the option –ksp_typedcg. Once the operators, right hand side, tolerances and other options are set, we have to set the vector for the deflation subspace construction by KSPDCGSetDeflation(). The KSPSolve_DCG() function is then based on KSPSolve_CG() calling after each search direction update the function KSPDCGApplyDeflation() solving the coarse problem and modifying this search direction.
To use MAGMA in the offload mode –ksp_cp_type cp_magma has to be specified. KSPDCGApplyDeflation() then sends right hand side vector from host to device, solves the coarse problem and sends the solution vector back from device to host.
The described KSP DCG type is available from FLLOP library [5] based on PETSc [4]. To use the solver with the FMPS code one can either include entire FLLOP library libfllop.h or compile FMPS with two additional files kspdcg.c and dcgimpl.h.

4. Conclusions

As a conclusion to the work on the hybrid MPI/OpenMP solver, porting communication intensive MPI applications to a Intel Xeon Phi cluster might yield quite a different scaling behaviour compared to a CPU only cluster, if more than one Xeon Phi card is used, although the shared memory part has been optimised for Xeon Phi. This is probably due to a message transport over the PCIe 2.0 bus.

The implemented DCG method reduces significantly the number of CG iterations and therefore also computational times. Further time savings can be reached using MIC accelerators, the reduced time is proportional to the number of solves we perform.

References

[1] A.Monakov, A.Lokhmotov, and A.Avetisyan (2010). Automatically Tuning Sparse Matrix-Vector Multiplication for GPU Architectures,In HiPEAC 2010, 111 - 125.
[2] E. Cuthill and J. McKee (1969). Reducing the bandwidth of sparse symmetric matrices, In Proc. 24th Nat. Conf. ACM, 157 - 172.
[3] M. Barth, M. Byckling, N. Ilieva, S. Saarinen, M. Schliephake, V. Weinberg (2013). Best Practice Guide - Intel Xeon Phi, v0.1: http://www.prace-ri.eu/Best-Practice-Guide-Intel-Xeon-Phi-HTML
[4] http://www.mcs.anl.gov/petsc/
[5] http://industry.it4i.cz/produkty/fllop/
[6] http://icl.cs.utk.edu/magma/software/
[7] http://crd-legacy.lbl.gov/~xiaoye/SuperLU/
[8] https://hpcforge.org/plugins/mediawiki/wiki/elmer/index.php/Docs

Acknowledgements

This work was financially supported by the PRACE-1IP project funded in part by the EUs 7th Framework Programme (FP7/2007-2013) under grant agreement no. RI-261557.
This publication was also supported by the Projects of major infrastructures for research, development and innovation of Ministry of Education, Youth and Sports (LM2011033), the European Regional Development Fund in the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070). Special thanks belong to our colleague Tomáš Brzobohatý from VSB who generated data for numerical experiments.